REMARKS

Claim 14 is amended. Claims 1-47 remain in the application for consideration. In view of the following remarks, Applicant respectfully requests the application be forwarded to issuance.

Drawings

The drawings have been objected to for having unacceptable margins. Particularly, Figs. 1 and 5 are indicated to have unacceptable left margins. Applicant submits herewith new figures having appropriate margins thus addressing the Office's objections.

The Specification

The Abstract section is objected to as having too many words. Applicant hereby amends its Abstract section to have an appropriate number of words.

In addition, the disclosure is objected to as containing an embedded hyperlink. Applicant has amended the specification to remove the hyperlink thereby overcoming the objection.

Further, the disclosure is objected to as not having a start tag associated with the closing tag "</contentlength>". Applicant has corrected this informality and thanks the examiner for the examiner's attention to detail.

§112 Rejections

Claim 22 stands rejected under §112, first paragraph as containing subject matter which is, in the Office's opinion, not enabled. The Office contends that the specification does not disclose a method for processing the XML document

without building the whole tree. Applicant respectfully disagrees and traverses the Office's rejection.

Claim 22 depends from claim 21 which, in turn, depends from claim 20. Claim 22 recites that the act of sending the response portion comprises doing so without building an entire hierarchical tree structure that represents an entire response for the client's request.

Applicant respectfully directs the Office's attention to page 18's subsection entitled "XML Response Generator". This section describes but one way in which the subject matter of claim 22 can be practiced. For the Office's convenience, this section of the specification is provided in its entirety just below:

XML Response Generator

To reduce processing overhead complexities and increase client response efficiencies, an XML response generator is provided that builds and sends portions of a client response to a client one piece at a time. This enables the client to begin processing the response so that the data contained therein can be put to use in a more timely fashion. *Accordingly, the piecewise processing and sending of the client response portions renders it unnecessary for an entire hierarchical tree structure to be built and stored in memory prior to generating the XML response.*

Fig. 7 shows an exemplary implementation of an XML response generator 300 that is configured for use in connection with Microsoft's Internet Information Service (IIS). The response generator 300 includes one or more request method objects 302, 304, and 306. *There is one request method object for each of the client request types that might be received.* For example, for a PROPFIND request, a request method object "CpropFindRequest" is created. The XML response generator also includes an emitter object 308 and a body object 310. *In this example, the request method objects 302, 304, and 306 together with the emitter object 308 constitute response-preparing mechanisms that prepare only portions of a response at a time.* A buffer 312 is provided and, in this example, is associated with the body object 310. Buffer 312 buffers response portions that are received from the emitter object 308. The buffer has a defined threshold which, when satisfied, enables the body object 310 to send the

buffered response portions to the client. The buffered response portions that are sent at any one time constitute less than an entirety of a complete client response. *Accordingly, the body object 310 and buffer 312 cooperate to send the response portions that are received from the emitter object 308 to the client.* In this particular example, an Internet Service Application Programming Interface (ISAPI) extension 314 is provided and is a request-receiving mechanism that receives client XML requests. ISAPI extension 314 is also responsible for returning the piece-wise generated XML response data into IIS responsive to calls that the body object 310 makes to the ISAPI extension.

Fig. 8 shows a flow diagram that describes steps in a method for responding to an XML client request. When a client request is received, e.g. by the ISAPI extension 314 (Fig. 7), step 400 determines the type of client request. This can be determined from the request method, i.e. HTTP verb, that is used in the request. In the above example, a client request in the form of a WebDAV PROPFIND request is received. *When the request type is determined, step 402 creates or instantiates a request-type or request method object (e.g. objects 302, 304, or 306 in Fig. 7) for that particular request type.* In this example, there is a request-type object that can be instantiated for each particular request that might be received from a client. So, in this example, for a PROPFIND request a "CPropFindRequest" object is created.

Each request-type or request method object is aware of the information that is necessary for its particular type of response to be generated. In addition, each request type object knows the specific or defined order for the calls that it must make to the emitter object 308 (Fig. 7). That is, because the client response is being generated in a piecewise fashion without building the entire hierarchical tree structure for the response, each request type object needs to make sure that the order of the hierarchy is preserved in the individual response portions that are generated. For example (see Fig. 6), each "response" node can include only one "href" node. The "href" node must come before any "propstat" nodes. Each "propstat" node includes one "status" node and one "prop"node. In the past where an entire hierarchical tree structure was built in memory, nodes or elements could be added at any time and at any place in the tree because the response had not yet been formulated and sent to the client. Here, however, individual portions of the response are being generated and sent to the client so that a node or element cannot later be added to a response portion that has already been sent to the client. *The way that the order of the hierarchy is preserved is by generating and sending calls to the emitter object 308 in a defined order that ensures that*

the hierarchical nature of the overall response is preserved. So, using the tree structure of Fig. 6 as an example, the request type object would want to make calls in an order that results in generating: the "multistatus" element, then the first "response" element, then the "href" element for the first response element, then the first "propstat" element for the first "response" element, then the "status" element for the first "propstat" element, then the "prop" element for the first "propstat" element, and then the sub-elements under the "prop" element, and so on. Since the response portions are being generated and sent in a serial, piecewise fashion, it is important to preserve the hierarchical nature of the client's complete response. So, each request type object knows the information or data that it needs and the order of the calls it needs to make to the emitter object.

Once the request-type object has been created and gathers the information that is necessary for building a portion of the response, the request-type object calls the emitter object (step 404) and provides the information to the emitter object. The emitter object 308 then takes the information provided to it and generates syntactically appropriate XML response portions. In doing so, the emitter object builds the XML response in a piecewise, node-by-node fashion. When the emitter object has built an XML response portion, it emits that response portion to the body object 310 (step 406). In this example, the body object 310 manages a buffer 312. The buffer 312 has a set threshold that defines how much XML data it can hold. This enables the body object 310 to accumulate or collect XML data (step 408). Step 410 determines whether the buffer threshold has been reached. If the threshold has not been reached or satisfied, step 410 loops back to step 406 which emits additional response portions to the body object. *If, on the other hand, the buffer threshold has been reached, then step 412 sends the buffered XML response portions to the client.* In this example, the body object 310 calls the ISAPI extension 314 which then returns the XML data to IIS. Step 414 checks to see whether the client response is complete. If it is not, then step 414 loops back to step 406 which emits XML data from the emitter object 308 to the body object 310. If the response is complete, however, then the processing for that response is over (step 416).

An advantage of the described embodiment is that processing of a client's XML request does not require building an entire hierarchical tree structure in memory prior to preparing the XML response and sending it to the client. Rather, client responses are generated in a piecewise, serial fashion. Individual response portions are prepared and sent to the client as the portions are generated. This can assist the client in beginning its processing of a response that might in some instances be quite lengthy. In

addition, response processing advantages are achieved by separating functionalities into data-gathering functions that are directed to gathering data that is specific to a particular client request that is received, and data-formatting functions that format the data into syntactically correct XML response portions. (emphasis added).

Applicant submits that the above passage is sufficient description so as to enable one of skill in the art to practice an embodiment covered by the claim. Accordingly, the Office's rejection is traversed.

Claims 16, 17, 23, 25, 26, 41 and 42 stand rejected as omitting essential steps with such omission amounting to a gap between the steps. Applicant disagrees with the Office and submits that no essential steps are missing. The context and meaning of the claims, in light of the specification, is sufficient such that there is no gap or missing steps. As an example, consider claim 16 in combination with independent claim 14 from which it depends, both of which are provided below for the convenience of the Office:

> 14.   A method of responding to an Extensible Markup Language (XML) request comprising:
> receiving an XML request from a client;
> gathering data that is to appear in a response to the client's request;
> calling an emitter object and passing the emitter object the gathered data;
> formatting the gathered data into an appropriate XML syntax with the emitter object; and
> emitting formatted data from the emitter object.
>
> ***
>
> 16.   The method of claim 14, wherein:
> said calling comprises calling the emitter object multiple times; and
> said emitting comprises emitting multiple amounts of formatted data.

Note that the act of calling recited in claim 14 recites "calling an emitter object and passing the emitter object the gathered data". Claim 16 further defines the act of calling as "calling the emitter object multiple times." Because claim 14 recites that the emitter object is called and then passed the gathered data, it is apparent that one of the reasons the emitter object is called is to pass it the gathered data. Accordingly, claim 16's further recitation that the emitter is called multiple times, taken in concert with claim 14, establishes that the emitter object is passed gathered data responsive to the individual calls that are made to it. Similarly, by virtue of the fact that claim 14 recites formatting the gathered data with the emitter object and emitting the formatted data from the emitter object, it follows that claim 16's act of "emitting multiple amounts of formatted data" is directed to emitting data that was formatted by the recited formatting step of claim 14. Since this formatting step formats the gathered data and, in claim 16's context, this gathered data results from the multiple calls that are made to the emitter object, there is no omission of any essential step.

Accordingly, Applicant traverses the Office's rejection and declines to amend the above-mentioned claims as there is no omitted essential step.

§103 Rejections

Claims 1-11, 13, 16-18, 20-26, 30-35, 38, 41, 42, 44-47 stand rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,012,098 to Bayeh et al. (hereinafter "Bayeh") in view of U.S. Patent No. 6,249,844 to Schloss et al. (hereinafter "Schloss").

Claims 14, 19, and 37 stand rejected under §103(a) over Bayeh.

Claims 12, 15, 27-29, 36, 39, 40, and 43 stand rejected under §103(a) over Bayeh and Schloss in view of U.S. Patent No. 6,366,947 to Kavner.

Applicant disagrees with the Office's rejections of these claims and, in view of the discussion below, respectfully traverses the rejections.

Before undertaking a discussion of the Office's application of Bayeh and Schloss to the claims, the following discussion of the Bayeh reference is provided to aid in an appreciation of some of the differences between the subject matter described in the cited reference and the various claimed embodiments.

The Bayeh Reference

Bayeh is directed to a method and system that utilizes servlet pairing for isolation of the retrieval and rendering of data. In accordance with its disclosure, data retrieval logic is isolated to a data servlet, and presentation formatting is isolated to a rendering servlet. Servlet chaining is used to send the output of the data servlet to the rendering servlet. The data servlet formats its output data stream for transfer to a downstream servlet. This data stream is formatted using a language such as the Extensible Markup Language (XML), according to a specific Document Type Definition (DTD). The rendering servlet parses this XML data stream, using a style sheet that may be written using the Extensible Style Language (XSL), and creates a HyperText Markup Language (HTML) data stream as output.

Bayeh describes the processing that takes place on the server side as follows:

> At Step 230, the server which received the client's request routes it to the proper data servlet. ***

Steps 240 through 270 are implemented by a data servlet according to the present invention. These steps represent the logic associated with data retrieval, as well as minimal formatting of the data for transfer to a rendering servlet: the formatting is not a presentation format.

At Step 240, the data servlet processes the client request. The request will typically require retrieving data from some database available to the data servlet. The data servlet will format a database query request, using an appropriate query language that will depend on the type of database on which the relevant data is stored. ***

Step 250 is included to indicate that, optionally, additional processing may be performed on the data received in response to the database query. ***

At Step 260, the data servlet formats the database information as an XML data stream. As previously discussed, a DTD is used in this formatting step. The DTD specifies how specific predefined "tags" are to be inserted into the XML data stream. A tag is a keyword that identifies what the data is which is associated with the tag, and is typically composed of a character string enclosed in special characters. "Special characters" means characters other than letters and numbers, which are defined and reserved for use with tags. Special characters are used so that a parser processing the data stream will recognize that this a tag. A tag is normally inserted preceding its associated data: a corresponding tag may also be inserted following the data, to clearly identify where that data ends. ***

When the entire XML data stream required for the database results has been formatted by the data servlet, that data stream is sent on to the next servlet in the chain at Step 270. In the preferred embodiment, the next servlet is the rendering servlet.

Steps 280 through 320 are implemented by a rendering servlet according to the present invention. These steps represent the logic associated with data presentation formatting.

Step 280 receives the XML data stream created by, and sent by, the data servlet. Techniques for receiving a data stream from a chained servlet are well known in the art, and Step 280 is shown for completeness of depicting the function of the rendering servlet.

According to the preferred embodiment, the rendering servlet *must* parse the XML data stream, and reformat it into HTML. *This is necessary* because browsers, by convention, expect to receive data that has been formatted with HTML. As discussed previously, this parsing process requires two types of data input: the XML data stream, and style sheet information. In the preferred embodiment, an XSL style sheet is used. Techniques for writing parsers are well known in the art, and will not be described in detail herein.

Step 290 indicates that the rendering servlet locates the XSL style sheet. Typically, the style sheet will be stored as a file on a medium, such as a disk drive, accessible to the rendering servlet. Alternatively, the style sheet might be incorporated directly into the code of the rendering servlet; however, because incorporating the information directly into the code makes revising the style sheet more difficult, this alternative is less desirable than isolating the style sheet as a separate file.

At Step 300, the rendering servlet parses the XML data stream, using the input sources described above. A parser reads a data stream, looking for predefined strings of characters that the parser recognizes, and which indicate to the parser what type of data is represented. In the preferred embodiment, the DTD used in the data servlet specified predefined strings that were used as tags, as described above, and inserted into the XML data stream by the data servlet. The parser looks for these tags as it reads the XML data stream. When a recognized tag is found, the parser knows what type of XML document element appears in the data stream between this tag and its corresponding ending tag (or following this tag, if ending tags are not required). As the parser in the rendering servlet determines what each document element is, it creates a new data stream, formatted using HTML. The parser may also insert presentation style attributes into the HTML data stream, where the appropriate attribute to use is determined according to the type of document element the parser is currently processing. Creation of the HTML data stream is represented in FIG. 5 as Step 310, although one skilled in the art will recognize that the functions of Steps 300 and 310 are intermingled. That is, as the parser processes portions of the input XML data stream, it creates the corresponding HTML data stream, then processes another portion of the input data stream, creates more of the output data stream, etc., until the entire input data stream has been processed.

When the reformatted data stream is complete, Step 320 sends that HTML data stream back to the client's browser, to be processed by the browser for presentation to the user.

The Claims Rejected Over Bayeh and Schloss

Claim 1 recites a method of generating an XML document. In accordance with the recited method, only a portion of an XML document is prepared and sent to a client. The acts of preparing and sending are repeated until an entire XML document is sent to a client.

In making out the rejection of this claim, the Office notes that Bayeh discloses sending an HTML document to a client and that it would be obvious to replace the HTML document with an XML document. There appears to be no support in Bayeh whatsoever for the Office's contention that this would be an obvious replacement. In fact, Bayeh teaches directly away from any such replacement by specifically teaching that the XML data stream *must be* reformatted into an HTML stream and that this is *necessary*. See, e.g. column 11, lines 34-38.

In addition, the Office notes that Bayeh is silent on preparing only a portion of the XML [sic: HTML] and then repeating the steps. Applicant disagrees. Bayeh is not silent. Rather, Bayeh specifically teaches away from the concept of preparing only a portion of a document and sending only the portion to a client and then repeating the steps. Specifically, the Office's attention is directed to column 12, lines 13-15 which is provided just below:

When the reformatted data stream is complete, Step 320 sends that HTML data stream back to the client's browser, to be processed by the browser for presentation to the user.

That is, Bayeh is describing the processing that takes place after the HTML stream corresponding to the client's request has been processed. As Bayeh

specifically states, only *after* the reformatted data stream is complete does the stream get sent to the client. Accordingly, Bayeh teaches directly away from the subject matter of claim 1. The Office then cites to Schloss and argues that it teaches that an XML document can be split up into segments and processed. Based on Schloss's teaching, the Office argues that it would have been obvious to modify Schloss into Bayeh, as splitting data into segments and sending then all was a common optimization in network transport. Applicant respectfully but strongly disagrees.

First, there is no basis in Bayeh to make any of the modifications the Office suggests. Rather, there are specific teachings in Bayeh that teach directly away from the Office's suggested modifications. Second, Schloss simply describes the notion that an XML document can be parsed based on its tags. Specifically, the passage cited by the Office (column 3, lines 21-49) is set out in its entirety below:

> In a preferred embodiment, an XML-like document will be used as an example of a document described using some formal language, such as a markup language. FIG. 3 shows an example of an XML-like document. The key point here is that the document includes multiple segments (330), where each segment (330) is enclosed between a "start-tag" (310) and an "end-tag" (320). For example, "<cml: molecule>"(similarly, "<m: order>" and "<db: price>") is a start-tag (310) and its corresponding "end-tag" (320) is "</cml: molecule>" ("</m: order>" and "</db: price>", respectively). As depicted, the segments may be nested. For example, the segment with the <price> start-tag is included within the segment with the <m: order> start-tag. Thus parsing the document to recognize the segments can be done by matching each "end-tag" with the corresponding "start-tag", which is the first preceding "start-tag" of the same type at the same nested level. In markup languages such as XML, each segment can have a DTD (document type definition) to describe the semantics of the markup. It is an object of the present invention to select a subset of the segments contained in a document and recognize them as persistent object fragments. Fragment creation eligibility criterion will be introduced next to determine when an object fragment should be created. In the preferred embodiment, two sets of

creation eligibility criterion are considered. For each persistent object fragment, a persistent identity or name is assigned and tracked so that if the object fragment appears in multiple objects or multiple times in the same object, it will be recognized as the same fragment.

Hence, Schloss simply describes the notion that an XML document can be parsed. This teaching falls far short of the teaching ascribed by the Office. Accordingly, for at least this reason, claim 1 is allowable.

Claims 2-4 depend either directly or indirectly from claim 1 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 1, are neither shown nor suggested in the references of record, either singly or in combination with one another.

Claim 5 recites a method of responding to an Extensible Markup Language (XML) request. In accordance with the recited method, an XML request is received from a client. Only a portion of a response to the request is prepared. The response portion is recited to be sent to the client. As noted above, Bayeh neither discloses nor suggests preparing and sending only a portion of a response to a client. Schloss adds nothing of significance to this claim's rejection. Accordingly, claim 5 is allowable.

Claims 6-13 depend either directly or indirectly from claim 5 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 5, are neither shown nor suggested in the references of record, either singly or in combination with one another. Additionally, claim 12 is further

rejected over the Kavner reference (U.S. Patent No. 6,366,947). Given Bayeh's shortcomings, this reference adds nothing of significance to claim 12's rejection.

Claim 14 recites a method of responding to an Extensible Markup Language (XML) request. In accordance with the recited method, an XML request is received from a client. Data that is to appear in a response to the client's request is gathered. An emitter object is called and passed the gathered data. The gathered data is formatted into an appropriate XML syntax with the emitter object and emitted from the emitter object. In addition, this claim has been amended to clarify that the emitter object emits the formatted data in a manner in which an XML response can be sent to the client without having to build a hierarchical tree that represents the XML response.

In making out the rejection, the Office argues that Bayeh discloses receiving a request and cites to column 10, lines 19-25 in support therefore. While Bayeh does disclose receiving a client request, it does not disclose receiving an XML request from the client. More importantly, Bayeh specifically discloses that it is necessary to send an HTML data stream back to the client's browser. (See, e.g. column 12, lines 13-15). Thus, Bayeh's response is an HTML response and *not* an XML response. Even more importantly, this claim has been amended to recite that the emitter object emits the formatted data in a manner in which an XML response can be sent to the client *without having to build a hierarchical tree* that represents the XML response. Accordingly, as Bayeh neither discloses nor teaches the subject matter of this claim, this claim is allowable.

Claims 15-19 depend directly from claim 14 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 14, are

neither shown nor suggested in the references of record, either singly or in combination with one another. In view of the allowability of these claims, the references to Schloss and Kavner are not seen to add anything of significance to those claims rejected in view of these references.

Claim 20 recites a method of responding to an Extensible Markup Language (XML) request. In accordance with the recited method, an XML request is received from a client and contains a Web Distributed Authoring and Versioning (WebDAV) request method. As noted above, Bayeh does not disclose receiving any such request. The method further recites determining the WebDAV request method that is contained in the client's request. Bayeh is absolutely silent and does not even suggest determining a WebDAV request method that is contained in the client's request. The method further recites creating a request method object for the WebDAV request method. As Bayeh does not disclose or suggest determining a WebDAV request method, it is virtually impossible for it to disclose or suggest creating a request method object for the WebDAV request method. Accordingly, for at least these reasons, this claim is allowable.

The claim further recites gathering data that is to appear in a response to the client's request with the request method object, calling an emitter object and passing the emitter object data that was gathered by the request method object, and generating at least a portion of a syntactically correct XML response with the emitter object using the data that was gathered by the request method object.

As Bayeh neither discloses nor suggests creating a request method object for the WebDAV request method, it is virtually impossible for it to disclose or suggest gathering data that is to appear in a response to the client's request with the request method object, calling an emitter object and passing the emitter object

data that was gathered by the request method object. Accordingly, for at least these additional reasons, this claim is allowable.

The Office notes that Bayeh is silent as to generating a portion of the response. The claim, however, recites generating at least a portion of a syntactically correct XML response. As Bayeh's responses are HTML responses by necessity, Bayeh teaches directly away from generating at least a portion of a syntactically correct XML response. Accordingly, for at least this additional reason, this claim is allowable. The Office's reliance on Schloss for its teaching that an XML document can be split up into segments and processed is misplaced. Schloss simply describes that XML documents can be parsed, which is a common feature of XML documents. Such teaching has no bearing or relevance with respect to the entirety of claim 20's subject matter.

Accordingly, for all of these reasons, claim 20 is allowable.

Claims 21-30 depend directly or indirectly from claim 20 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 20, are neither shown nor suggested in the references of record, either singly or in combination with one another. In view of the allowability of these claims, the references to Schloss and Kavner are not seen to add anything of significance to those claims rejected in view of these references.

Claim 31 recites an Extensible Markup Language (XML) request processor. The XML response generator is recited to comprise a request-receiving mechanism configured to receive an XML request from a client. A response-preparing mechanism is coupled with the request-receiving mechanism and configured to prepare *only a portion of a response at a time*. A sending

mechanism is coupled with the response-preparing mechanism and is configured to receive response portions from the response-preparing mechanism and to send the response portions to the client. The sent response portions are recited to constitute less than an entirety of a response.

As noted above, Bayeh does not disclose an XML request processor. In addition, the Office states that Bayeh is silent on preparing only a portion of an XML response. This is incorrect. Bayeh specifically teaches that it prepares a *complete response* and once the complete response is prepared, it is sent to the client. (See, e.g. column 12, lines 13-15). Accordingly, Bayeh teaches directly away from the recited subject matter. Schloss adds nothing of significance to this rejection as Schloss teaches only that XML documents can be parsed. Accordingly, this claim is allowable.

Claims 32-36 depend directly from claim 31 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 31, are neither shown nor suggested in the references of record, either singly or in combination with one another. In view of the allowability of these claims, the references to Schloss and Kavner are not seen to add anything of significance to those claims rejected in view of these references.

Claim 37 recites an Extensible Markup Language (XML) request processor. The XML request processor comprises a data-gathering object for gathering data that is to appear in a client response and generating calls *in a predefined order* that contain the gathered data. An emitter object is recited to be configured to receive calls that are generated by the data-gathering object and format the data contained therein into an appropriate XML syntax.

Nowhere does Bayeh disclose or suggest any such subject matter. Specifically, nowhere does Bayeh disclose or suggest a data-gathering object for generating calls *in a predefined order* that contain data that has been gathered by the data-gathering object. Accordingly, for at least this reason, this claim is allowable.

Claims 38-40 depend either directly or indirectly from claim 37 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 37, are neither shown nor suggested in the references of record, either singly or in combination with one another. In view of the allowability of these claims, the references to Schloss and Kavner are not seen to add anything of significance to those claims rejected in view of these references.

Claim 41 recites a computer-readable medium having a computer program for responding to an XML request. The program is recited to comprise the following steps:

- receiving a client request;
- *determining an HTTP verb* that is contained in the client request;
- instantiating a request method object that *corresponds to* the HTTP verb that is contained in the client request;
- using the request method object to gather information that is to appear in a response to the client's request;
- making a series of calls to an emitter object that is configured to receive information from the request method object and process the information into a *response portion* having an appropriate XML syntactic format; and
- *sending the response portion* to the client.

The Office argues that Bayeh discloses receiving a request and a servlet that is equivalent to an object for gathering a response to the request. The Office

then argues that this is equivalent to an object that corresponds to an HTTP verb and that it would have been inherent to determine the particular method before processing the request. Applicant respectfully but strongly disagrees. It is inappropriate for the Office to assume that a reference possesses inherent characteristics in an attempt to bridge a gap in the reference. The claim specifically recites *determining an HTTP verb* that is contained in the client request and instantiating a request method object that *corresponds to* the HTTP verb that is contained in the client request. Bayeh neither discloses nor suggests these features and it is inappropriate for the Office to assume that Bayeh inherently embodies characteristics that it does not. There is simply no support whatsoever for the Office's assertion that Bayeh inherently determines an HTTP verb and then instantiates a request method object that *corresponds* to the HTTP verb, as that term is used in Applicant's specification.

The Office further contends that it would be obvious to send Bayeh's unconverted XML data stream to the client rather than the HTML stream. This assertion completely ignores Bayeh's specific teaching that it is necessary to convert the XML data stream to an HTML data stream. Thus, the Office has impermissibly modified Bayeh without any support in Bayeh for such modification.

Further, the Office argues that it would be obvious to modify Bayeh, using Schloss's teachings, to call the emitter multiple times. Applicant disagrees. Claim 41 recites making a series of calls to an emitter object which is configured to receive information and then process the information into a response portion that is sent to the client. Bayeh specifically teaches that its HTML response must be

complete before it is sent to the client. As such, Bayeh teaches directly away from sending a response portion to the client.

Accordingly, for all of these reasons, Claim 41 is allowable.

Claims 42 and 43 depend directly from claim 41 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 41, are neither shown nor suggested in the references of record, either singly or in combination with one another. In view of the allowability of these claims, the references to Schloss and Kavner are not seen to add anything of significance to those claims rejected in view of these references.

Claim 44 recites a computer-readable medium having software code that is configured to receive an XML request from a client and *instantiate an object that corresponds to an HTTP verb* that is contained in the request. The software code further uses the object to build a portion of an XML response to the request.

As noted above, Bayeh neither discloses nor suggests software code that receives an XML request. Further, Bayeh neither discloses nor suggests software code that instantiates an object that *corresponds* to an HTTP verb that is contained in the request, as that term is used in Applicant's specification. Accordingly, for at least this reason, this claim is allowable. The Office's reliance on Schloss is not seen to add anything of significance to the Office's rejection of this claim.

Claims 45-47 depend directly from claim 44 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 44, are neither shown nor suggested in the references of record, either singly or in combination with one another.

Conclusion

The claims are in condition for allowance and Applicant respectfully requests that the application be forwarded on to issuance. If the Office's next anticipated action is to be anything other than issuance of a Notice of Allowability, the undersigned respectfully requests a telephone call for the purpose of scheduling an interview.

Version of the Specification with Markups to Show Changes

On page 17, starting at line 10 through page 18, line 2, please replace the text with the following:

```
<?xml version="1.0" ?>
<a:multistatus xmlns:a="DAV:"
xmlns:b="urn:uuid:c2f41010-65b3-11d1-a29f-00aa00c14882/" >
<a:response>
        <a:href>[http://server/folder/file1.html] http:_server_folder_file1.html </a:href>

        <a:propstat>
        <a:status>HTTP/1.1 200 OK</a:status>
        <a:prop>
                <a:getcontentlength b:dt="int" >694</a:contentlength>
                <a:getcontenttype>text/html</a:getcontenttype>
        </a:prop>
        </a:propstat>
        <a:propstat>
        <a:status>HTTP/1.1 404 Resource Not Found</a:status>
        <a:prop>
                <author/>
        </a:prop>
        </a:propstat>
</a:response>

<a:response>
        <a:href>http://server/folder/test2.html</a:href>
        <a:propstat>
        <a:status>HTTP/1.1 200 OK</a:status>
        <a:prop>
                <a:getcontentlength b:dt="int" >1064</a:getcontentlength>
                <a:getcontenttype>text/html</a:getcontenttype>
        </a:prop>
        </a:propstat>
        <a:propstat>
        <a:status>HTTP/1.1 404 Resource Not Found</a:status>
                <a:prop>
                <author/>
        </a:prop>
        </a:propstat>
</a:response>
</a:multistatus
```

Version of the Claims with Markups to Show Changes

14. (Amended) A method of responding to an Extensible Markup Language (XML) request comprising:

receiving an XML request from a client;

gathering data that is to appear in a response to the client's request;

calling an emitter object and passing the emitter object the gathered data;

formatting the gathered data into an appropriate XML syntax with the emitter object; and

emitting formatted data from the emitter object, the emitter object emitting the formatted data in a manner in which an XML response can be sent to the client without having to build a hierarchical tree that represents the XML response.

Respectfully Submitted,

Dated: 12/17/02
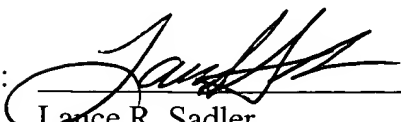
By: _____
Lance R. Sadler
Reg. No. 38,605
(509) 324-9256